As you all know, in Java, a class provides the blueprint for objects, you create an object from a class. There are many different ways to create objects in Java.  
**Following are some ways in which you can create objects in Java:**

**1) Using new Keyword :** Using new keyword is the most basic way to create an object. This is the most common way to create an object in java. Almost 99% of objects are created in this way. By using this method we can call any constructor we want to call (no argument or parameterized constructors).

|  |
| --- |
| // Java program to illustrate creation of Object  // using new keyword  public class NewKeywordExample  {      String name = "GeeksForGeeks";      public static void main(String[] args)      {          // Here we are creating Object of          // NewKeywordExample using new keyword          NewKeywordExample obj = new NewKeywordExample();          System.out.println(obj.name);      }  } |

Output:

GeeksForGeeks

**2) Using**[**New Instance**](https://www.geeksforgeeks.org/new-operator-vs-newinstance-method-java/)**:**If we know the name of the class & if it has a public default constructor we can create an object –**Class.forName**. We can use it to create the Object of a Class. Class.forName actually loads the Class in Java but doesn’t create any Object. To Create an Object of the Class you have to use the new Instance Method of the Class.

|  |
| --- |
| // Java program to illustrate creation of Object  // using new Instance  **package** com.learning.serializable;  **public** **class** NewInstanceExample  {  String name = "GeeksForGeeks";  **public** **static** **void** main(String[] args)  {  **try**  {  Class cls = Class.*forName*("com.learning.serializable.NewInstanceExample");  NewInstanceExample obj =  (NewInstanceExample) cls.newInstance();  System.***out***.println(obj.name);//Accessing variable with class name  }  **catch** (ClassNotFoundException e)  {  e.printStackTrace();  }  **catch** (InstantiationException e)  {  e.printStackTrace();  }  **catch** (IllegalAccessException e)  {  e.printStackTrace();  }  }  } |

Output:

GeeksForGeeks

**3) Using**[**clone() method:**](https://www.geeksforgeeks.org/clone-method-in-java-2/) Whenever clone() is called on any object, the JVM actually creates a new object and copies all content of the previous object into it. Creating an object using the clone method does not invoke any constructor.  
To use clone() method on an object we need to implement **Cloneable** and define the clone() method in it.

|  |
| --- |
| // Java program to illustrate creation of Object  // using clone() method  **package** com.learning.serializable;  **public** **class** CloneExample **implements** Cloneable  {  @Override  **protected** Object clone() **throws** CloneNotSupportedException  {  **return** **super**.clone();  }  String name = "GeeksForGeeks";    **public** **static** **void** main(String[] args)  {  CloneExample obj1 = **new** CloneExample();  **try**  {  CloneExample obj2 = (CloneExample) obj1.clone();  System.***out***.println(obj2.name);  }  **catch** (CloneNotSupportedException e)  {  e.printStackTrace();  }  }  } |

Output:

GeeksForGeeks

**Note :**

* Here we are creating the clone of an existing Object and not any new Object.
* Class need to implement Cloneable Interface otherwise it will throw **CloneNotSupportedException**.

**4) Using**[**deserialization**](https://www.geeksforgeeks.org/serialization-in-java/)**:** Whenever we serialize and then deserialize an object, JVM creates a separate object. In **deserialization**, JVM doesn’t use any constructor to create the object.  
To deserialize an object we need to implement the Serializable interface in the class.

**Serializing an Object :**

|  |
| --- |
| // Java program to illustrate Serializing  // an Object.  SerializeClass.java  **package** com.learning.serializable;  **import** java.io.Serializable;  **public** **class** SerializeClass **implements** Serializable{  **private** **static** **final** **long** ***serialVersionUID*** = 55L;    **private** String name;  **public** SerializeClass(String name) {  **this**.name=name;  }  **public** String getName() {  **return** name;  }  **public** **void** setName(String name) {  **this**.name = name;  }    } |

Object of DeserializationExample class is serialized using writeObject() method and written to file.txt file.

**Deserialization of Object :**

**package** com.learning.serializable;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FileNotFoundException;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.io.ObjectInputStream;

**import** java.io.ObjectOutputStream;

**public** **class** DeserializationExample {

**public** **static** **void** main(String[] args) {

//serialize();

*deserialize*();

}

**public** **static** **void** serialize() {

**try** {

SerializeClass serializeClass = **new** SerializeClass("GeeksForGeeks");

FileOutputStream fileOutputStream = **new** FileOutputStream("test.txt");

ObjectOutputStream objectOutputStream = **new** ObjectOutputStream(fileOutputStream);

objectOutputStream.writeObject(serializeClass);

objectOutputStream.close();

fileOutputStream.close();

System.***out***.println("Serialization completed");

} **catch** (FileNotFoundException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

}**catch** (Exception e) {

e.printStackTrace();

}

}

**public** **static** **void** deserialize() {

SerializeClass serializeClass = **null**;

**try** {

FileInputStream fileInputStream = **new** FileInputStream("test.txt");

ObjectInputStream objectInputStream = **new** ObjectInputStream(fileInputStream);

serializeClass = (SerializeClass) objectInputStream.readObject();

System.***out***.println("De Serializable Object property name "+serializeClass.getName());

} **catch** (FileNotFoundException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

} **catch** (ClassNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

}

Output:

GeeksForGeeks

**5) Using newInstance() method of Constructor class :** This is similar to the newInstance() method of a class. There is one newInstance() method in the **java.lang.reflect.Constructor** class which we can use to create objects. It can also call parameterized constructor, and private constructor by using this newInstance() method.

Both newInstance() methods are known as reflective ways to create objects. In fact newInstance() method of Class internally uses newInstance() method of Constructor class.

|  |
| --- |
| // Java program to illustrate creation of Object  // using newInstance() method of Constructor class  **package** com.learning.serializable;  **import** java.lang.reflect.\*;  **public** **class** ReflectionExample {  **private** String name;  ReflectionExample() {  }  **public** **void** setName(String name) {  **this**.name = name;  }  **public** **static** **void** main(String[] args) {  **try** {  Constructor<ReflectionExample> constructor = ReflectionExample.**class**.getDeclaredConstructor();  ReflectionExample r = constructor.newInstance();  r.setName("GeeksForGeeks");  System.***out***.println(r.name);  } **catch** (Exception e) {  e.printStackTrace();  }  }  } |

Output:

GeeksForGeeks